Shadows

Thanks to:
Frédo Durand
and Seth Teller
MIT

“Now this is... this is... well, I guess it's another snake.”

Shadows

- Realism
- Depth cue
Shadows as depth cue

Spatial relationship between objects
Spatial relationship between objects
Spatial relationship between objects

Michael McCool
Univ of Waterloo

Spatial relationship between objects

Michael McCool
Univ of Waterloo
Shadows add visual acuity

Shadows and art

- Only in Western pictures (here Caravaggio)
Shadows and art

• Shadows as the origin of painting

Duality shadow-view

• A point is lit if it is visible from the light source

• Shadow computation very similar to view computation
Shadow ray

- Ray from visible point to light source
- If blocked, discard light contribution
- One shadow ray per light
- Optimization?
  - Stop after first intersection (don’t worry about tmin)
  - Test latest obstacle first

Ray-casting shadows

Fig. 16.52 Early pictures rendered with ray tracing. (Courtesy of Arthur Appel, IBM T.J. Watson Research Center.)
Local vs. Global Illumination

• Core OpenGL uses local illumination model
  – Light sources used: distant, point, spot
  – Pros:
    • Fast calculation of pixel color using Phong, only needs
      – Position & direction of light source
      – Material properties (diffuse, specular, ambient coeff)
      – Light settings (intensity, fall-off, color)
      – …but no info about other objects in the scene!
    • Each primitive can be processed independently of others
  – Cons:
    • Good looking images need global effects
      – Reflected light (e.g. environment mapping)
      – Shadows

Global: Shadows

• A point is in shadow if the light got blocked between the light source and point

• Need mask that contains information about blocked / non blocked pixels
Fake methods

• Still commonly used in games

• Shadows are simple, hand-drawn polygons
• No global effect
  …but better than no shadow at all 😊

Images from TombRaider. ©Eidos Interactive.

Shadow Quality: “Blobs”
Overview

• Projected Shadows

• Shadow map
  – Image-precision, texture mapping

• Shadow volume
  – Object space

• Soft shadows

Planar Projection

• Render a ground-plane
• Render an object
• Then render the object again, but this time
  – Projected onto the plane
  – Without light, so that the shadow is black
  – Half transparent (using blending), to avoid completely dark shadows
  – Avoid multiple “darkening” on one spot by using ordinary z-buffer checks
Projected Geometry

• [Blinn88] *Me and my fake shadow*
  – Shadows for selected large receiver polygons
    • Ground plane
    • Walls

Projected Geometry

• Example: $xz$ plane at $y=0$

$$p_x = \frac{l_y v_x - l_x v_y}{l_y - v_y}$$

$$p_z = \frac{l_y v_z - l_z v_y}{l_y - v_y}$$
Projected Geometry

• Transformation as 4 by 4 matrix

\[
\tilde{p} = \begin{pmatrix}
l_y & -l_x & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & -l_z & l_y & 0 \\
0 & -1 & 0 & l_y
\end{pmatrix}
\begin{pmatrix}
v_x \\
v_y \\
v_z \\
1
\end{pmatrix}
\]

Projected Geometry

• General case: receiver polygon in plane \( E \)

\[
E : \vec{n} \cdot \vec{x} + d = 0
\]

\[
\tilde{p} = \vec{l} - \frac{d + \vec{n} \cdot \vec{l}}{\vec{n} \cdot (\vec{v} - \vec{l})}(\vec{v} - \vec{l})
\]

• 4x4 matrix (see Blinn)
Projected Geometry

• Basic algorithm
  – Render scene (full lighting)
  – For each receiver polygon
    • Compute projection matrix $M$
    • Mult with actual transformation (modelview)
    • Render selected (occluder) geometry
      – Darken/Black

Planar Shadows

Shadow is projected into the plane of the floor.
Constructing a Shadow Matrix

```c
void shadowMatrix(GLfloat shadowMat[4][4], GLfloat groundplane[4], GLfloat lightpos[4])
{
    GLfloat dot;
    /* Find dot product between light position vector and ground plane normal. */
    dot = groundplane[X] * lightpos[X] +
         groundplane[Y] * lightpos[Y] +
         groundplane[Z] * lightpos[Z] +
         groundplane[W] * lightpos[W];
    shadowMat[0][0] = dot - lightpos[X] * groundplane[X];
    shadowMat[1][0] = 0.f - lightpos[X] * groundplane[Y];
    shadowMat[2][0] = 0.f - lightpos[X] * groundplane[Z];
    shadowMat[3][0] = 0.f - lightpos[X] * groundplane[W];
    shadowMat[0][1] = 0.f - lightpos[Y] * groundplane[X];
    shadowMat[1][1] = dot - lightpos[Y] * groundplane[Y];
    shadowMat[2][1] = 0.f - lightpos[Y] * groundplane[Z];
    shadowMat[3][1] = 0.f - lightpos[Y] * groundplane[W];
    shadowMat[0][2] = 0.f - lightpos[Z] * groundplane[X];
    shadowMat[1][2] = 0.f - lightpos[Z] * groundplane[Y];
    shadowMat[3][2] = 0.f - lightpos[Z] * groundplane[W];
    shadowMat[0][3] = 0.f - lightpos[W] * groundplane[X];
    shadowMat[1][3] = 0.f - lightpos[W] * groundplane[Y];
    shadowMat[2][3] = 0.f - lightpos[W] * groundplane[Z];
}
```

How to add shadows?

- Can be done in two ways:
  - 1st method: Full illumination + darkening

  \[
  \text{FB} = \text{DiffuseTex0} \times (\text{Light0} + \text{Light1} + \text{Light2}…)
  \]
  \* if pixel is in shadow (with respect to Light0) \*

  \[
  \text{FB} = \text{FB} \times 0.5
  \]

  This is wrong since the contribution of Light1,2 etc. is also affected!
How to add shadows?

- 2nd & correct method: Use mask for each light

\[ FB = \text{DiffuseTex0} \times ( \text{Light0} \times \text{Mask0} + \text{Light1} \times \text{Mask1} + \text{Light2} \times \text{Mask2} \ldots ) \]

Mask values
- 0 if pixel is in shadow (with respect to Light X)
- 1 if pixel is lit by Light X
- 0…1 for pixels on shadow edge (soft shadow edge)

Accumulation of \((\text{Light0} \times \text{Mask0}) + \ldots\) can be done using additive blending

How to add shadows?

- Algorithm
  - Render scene with ambient illumination only
  - For each light source
    - Render scene with illumination from this light only
    - Scale illumination by shadow mask
    - Add up contribution to frame buffer

- Expensive but nearly correct!

- Speed-Up
  - Use more lights & masks in one pass
    - Masks stored as textures
    - Apply masks & sum up using e.g. register combiners
How to Render the Shadow

/* Render 50% black shadow color on top of whatever the floor appearance is. */
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,
          GL_ONE_MINUS_SRC_ALPHA);
glDisable(GL_LIGHTING); /* Force the 50% black. */
glColor4f(0.0, 0.0, 0.0, 0.5);

glPushMatrix();
/* Project the shadow. */
glMultMatrixf((GLfloat *) floorShadow);
drawDinosaur();
glPopMatrix();

Not Quite So Easy (1)

Without stencil to avoid double blending of the shadow pixels:

Notice darks spots on the planar shadow.

Solution: Clear stencil to zero. Draw floor with stencil of one. Draw shadow if stencil is one. If shadow’s stencil test passes, set stencil to two. No double blending.
Not Quite So Easy (2)

There’s still another problem even if using stencil to avoid double blending.

Shadow fights with depth values from the floor plane. Use polygon offset to raise shadow polygons slightly in Z.

Not Quite so Easy (3)

Good.

Notice right image’s reflection falls off the floor!

Same problem with Shadows!
Planar Projection

- Fast
  - Can be done with a matrix operation
- Easy
  - Just use the Modelview transform
- Very unrealistic
  - Just planar shadows

Projected Geometry

- Problems
  - Z-Fighting
    - Use bias when rendering shadow polygons
    - Use stencil buffer (no depth test)
  - Bounded receiver polygon?
    - Use stencil buffer (restrict drawing to receiver area)
  - Shadow polygon overlap?
    - Use stencil count (only the first pixel gets through)
Fake shadows using textures

- Separate obstacle and receiver
- Compute b/w image of obstacle from light
- Use projective textures

![Image from light source](image1.png) ![BW image of obstacle](image2.png) ![Final image](image3.png)

Figure from Moller & Haines “Real Time Rendering”

Fake shadows using textures

- Limitations?

![Image from light source](image4.png) ![BW image of obstacle](image5.png) ![Final image](image6.png)

Figure from Moller & Haines “Real Time Rendering”
Shadow maps

- Use texture mapping but using depth
- 2 passes (at least)
  - Compute shadow map from light source
    - Store depth buffer (shadow map)
  - Compute final image
    - Look up the shadow map to know if points are in shadow

Figure from Foley et al. “Computer Graphics Principles and Practice”

Shadow map look up

- We have a 3D point \(x, y, z\)
- How do we look up the shadow map?

Figure from Foley et al. “Computer Graphics Principles and Practice”
Shadow map look up

- We have a 3D point \( x, y, z \)
- How do we look up the shadow map?
- Use the 4x4 camera matrix from the light source
- We get \( (x', y', z') \)
- Test:
  \[ \text{ShadowMap}(x', y') < z' \]

Figure from Foley et al. “Computer Graphics Principles and Practice”

Shadow maps

- In Renderman
  - (High-end production software)
Shadow maps

- Can be done in hardware
- Using hardware texture mapping
  - Texture coordinates u,v,w generated using 4x4 matrix
  - Modern hardware permits tests on texture values

Introducing Another Technique: Shadow Mapping

- Image-space shadow determination
  - Lance Williams published the basic idea in 1978
    - By coincidence, same year Jim Blinn invented bump mapping
      (a great vintage year for graphics)
  - Completely image-space algorithm
    - means no knowledge of scene’s geometry is required
    - must deal with aliasing artifacts
  - Well known software rendering technique
    - Pixar’s RenderMan uses the algorithm
    - Basic shadowing technique for Toy Story, etc.
Shadow Mapping References

• Important SIGGRAPH papers
  – Lance Williams, “Casting Curved Shadows on Curved Surfaces,” SIGGRAPH 78
  – William Reeves, David Salesin, and Robert Cook (Pixar), “Rendering antialiased shadows with depth maps,” SIGGRAPH 87

The Shadow Mapping Concept (1)

• Depth testing from the light’s point-of-view
  – Two pass algorithm
  – First, render depth buffer from the light’s point-of-view
    • the result is a “depth map” or “shadow map”
    • essentially a 2D function indicating the depth of the closest pixels to the light
  – This depth map is used in the second pass
The Shadow Mapping Concept (2)

- Shadow determination with the depth map
  - Second, render scene from the eye’s point-of-view
  - For each rasterized fragment
    - determine fragment’s XYZ position relative to the light
    - this light position should be setup to match the frustum
      used to create the depth map
    - compare the depth value at light position XY in the depth
      map to fragment’s light position Z

The Shadow Mapping Concept (3)

- The Shadow Map Comparison
  - Two values
    - $A = Z$ value from depth map at fragment’s light XY position
    - $B = Z$ value of fragment’s XYZ light position
  - If $B$ is greater than $A$, then there must be
    something closer to the light than the fragment
    - then the fragment is shadowed
  - If $A$ and $B$ are approximately equal, the
    fragment is lit
**Shadow Mapping with a Picture in 2D (1)**

The $A < B$ shadowed fragment case

**Shadow Mapping with a Picture in 2D (2)**

The $A \equiv B$ unshadowed fragment case
Shadow Mapping with a Picture in 2D (3)

Note image precision mismatch!

The depth map could be at a different resolution from the framebuffer.
This mismatch can lead to artifacts.

Visualizing the Shadow Mapping Technique (1)

- A fairly complex scene with shadows

*the point light source*
Visualizing the Shadow Mapping Technique (2)

• Compare with and without shadows

*with shadows*  
*without shadows*

Visualizing the Shadow Mapping Technique (3)

• The scene from the light’s point-of-view

*F.Y.I.: from the eye’s point-of-view again*
Visualizing the Shadow Mapping Technique (4)

• The depth buffer from the light’s point-of-view

FYI: from the light’s point-of-view again

Visualizing the Shadow Mapping Technique (5)

• Projecting the depth map onto the eye’s view

FYI: depth map for light’s point-of-view again
Visualizing the Shadow Mapping Technique (6)

- Projecting light’s planar distance onto eye’s view

Green is where the light planar distance and the light depth map are approximately equal.

Non-green is where shadows should be.
Visualizing the Shadow Mapping Technique (7)

• Scene with shadows

Notice how specular highlights never appear in shadows

Notice how curved surfaces cast shadows on each other

Shadow Quality: Shadow Maps
Problems with shadow maps?

- Field of view
- Bias
- Aliasing

Field of view problem

- What if point to shadow is outside field of view of shadow map?
- Use cubical shadow map
- Use only spot lights!
Problems with shadow maps?

- Field of view
- Bias
- Aliasing

The bias nightmare

- For a point visible from the light source
  \[ \text{ShadowMap}(x',y') \approx z' \]
- Avoid erroneous self shadowing
The bias nightmare

- Shadow ray casting
  - Start the ray at hit + light * epsilon
  - Add bias to avoid degeneracy
  - Yet another instance of geometric robustness

Bias for shadow maps

ShadowMap(x', y') + bias < z'

Choosing the good bias value can be very tricky

- bias too small → surface acne
- bias too large → shadow leaks
Construct Light View Depth Map

• Realizing the theory in practice
  – Constructing the depth map
    • use existing hardware depth buffer
    • **use glPolygonOffset to bias depth value**
    • read back the depth buffer contents
  – Depth map can be copied to a 2D texture
    • unfortunately, depth values tend to require more precision than 8-bit typical for textures
    • Fortunately, we have floating-point textures
    • depth precision typically 16-bit or 24-bit

Justification for glPolygonOffset When Constructing Shadow Maps

• Depth buffer contains “window space” depth values
  – Post-perspective divide means non-linear distribution
  – glPolygonOffset is guaranteed to be a window space offset
• Doing a “clip space” glTranslatef is not sufficient
  – Common shadow mapping implementation mistake
  – Actual bias in depth buffer units will vary over the frustum
  – No way to account for slope of polygon
Sampling a Polygon’s Depth at Pixel Centers (1)

• Consider a polygon covering pixels in 2D

Sampling a Polygon’s Depth at Pixel Centers (2)

• Consider a 2nd grid for the polygon covering pixels in 2D
Sampling a Polygon’s Depth at Pixel Centers (3)

- How Z changes with respect to X

Why You Need glPolygonOffset’s Slope

- Say pixel center is re-sampled to another grid
  - For example, the shadow map texture’s grid!
- The re-sampled depth could be off by
  \(+/-0.5 \frac{\partial z}{\partial x}\) and \(+/-0.5 \frac{\partial z}{\partial y}\)
- The maximum absolute error would be
  \(|0.5 \frac{\partial z}{\partial x} + 0.5 \frac{\partial z}{\partial y}| \approx \max( |\frac{\partial z}{\partial x}|, |\frac{\partial z}{\partial y}| )\)
  - This assumes the two grids have pixel footprint area ratios of 1.0
  - Otherwise, we might need to scale by the ratio
- Exactly what polygon offset’s “slope” depth bias does
Depth Map Bias Issues

• How much polygon offset bias depends

Selecting the Depth Map Bias

• Not that hard
  – Usually the following works well
    • glPolygonOffset(scale = 1.1, bias = 4.0)
  – Usually better to error on the side of too much bias
    • adjust to suit the shadow issues in your scene
  – Depends somewhat on shadow map precision
    • more precision requires less of a bias
  – When the shadow map is being magnified, a larger scale is often required
Problems with shadow maps?

• Field of view
• Bias
• Aliasing

Shadow map aliasing

• Undersampling of shadow map
• Reprojection aliasing
Alaising

- Finite shadow map resolution
- Result: pixelized shadows

Shadow map filtering

- Does not work!
- Filtering depth is not meaningful

a) Ordinary texture map filtering. Does not work for depth maps.
Percentage closer filtering

• Filter the result of the test
• But makes the bias issue more tricky

Sample Transform Step

Percentage closer filtering

• 5x5 samples
• Nice antialiased shadow
• Using a bigger filter produces fake soft shadows
• But makes the bias issue more tricky
Shadows in production

- Often use shadow maps
- Ray casting as fallback in case of robustness issues

Movie Time!
Aliasing

• Bad aliasing cases:
  – Large Scenes
    • High resolution shadow map required
  – Close-ups to shadow boundaries
    • Zoom in
  – Shadow stretches along the receiver

Aliasing

• Duelling frusta
  – Light shines opposite to viewing direction
**Aliasing**

- Duelling frusta
  - Resolution mismatch

---

**Aliasing**

- Miner’s headlamp
  - Similar frusta
  - Similar sampling
  - One shadowmap pixel for image pixel
Hardware Support

• OpenGL 1.4
  – GL_ARB_DEPTH_TEXTURE
    internal texture format to store depth values
  – glTexGen
    generation of light space coordinates as texture coordinates
  – GL_ARB_SHADOW
    special texture mode:
    return texture(s,t) < r ? Black : White;

Hardware Support

• P-Buffers
  – offscreen rendering of shadow map
  – large shadow maps sizes

• GL_ARB_RENDER_TEXTURE and WGL_NV_RENDER_TEXTURE
  – bind P-Buffer depth buffer as depth texture
  – no copy needed
Hardware Support

• Register Combiners / Fragment Programs
  – for shadow application
  – if point is in shadow:
    • leave ambient light unchanged
      – ambient light has no origin, thus cannot be shadowed
    • dim diffuse light
      – some shading remains to show surface orientation
    • remove specular light
      – no highlights in shadow

Pros and Cons

+ general
  – everything that can be rendered can cast and receive a shadow
  – works together with vertex programs
+ fast
  – full hardware support
  – (almost) no overhead for static scenes
  – two passes needed for dynamic scenes
+ robust
+ easy to implement
- aliasing
Questions?
Overview

• Shadow map
  – Image-precision, texture mapping

• Shadow volume
  – Object space

• Soft shadows

Shadow volumes

• Explicitly represent the volume of space in shadow

• For each polygon
  – Pyramid with point light as apex
  – Include polygon to cap

• Shadow test similar to clipping
Shadow volumes naïve rendering

- Pick your favorite rendering algorithm
- For each visible point
  - For each light
    - For each shadow volume
      - If point inside volume
        » Point is in shadow

Great but costly
(#poly * #light tests for each visible point)

Shadow volumes smarter rendering

- New way to define inside/outside
- Consider ray from eye to visible point
- Increment or decrement a counter each time we intersect a shadow volume polygon
- Points lit: counter = 0
Hardware shadow volumes

- Add counter buffer
- Draw scene with no shading
- Turn off buffer update
- Draw frontfacing shadow polygons
  - If z-pass
  - Increment counter
- Draw backfacing shadow polygons
  - If z-pass
  - Decrement counter
- Compute shading
  - Lit Points have counter=0

Optimizing shadow volumes

- Use silhouette edges only
Problem if eye inside shadow

- Then a counter of 0 does not necessarily mean lit
- Three solutions
  - Explicitly test eye point wrt all volumes
  - Clip the shadow volumes and include these new polygons
  - Z-fail shadow volumes

Problem if eye inside shadow

- Clip the shadow volumes and include these new polygons
Z-fail shadow volume

- Count from infinity
- Draw scene with no shading
- Turn off buffer update
- Draw frontfacing shadow polygons
  - If z-fail
  - Decrease counter
- Draw backfacing shadow polygons
  - If z-fail
  - Increase counter
- Compute shading
  - Lit points have counter=0

Z-fail shadow volume

- Count from infinity
- Then we can have problems with far clipping plane
- Can be solved with new hardware trick
  - Depth clamp at clipping stage
  - Instead of clipping a polygon, it can be rendered with maximum depth
Questions?

Plate 52 Grandville, *The Shadows (The French Cabinet)* from *La Caricature*, 1830.